Exhibit 12 to Complaint
Intellectual Ventures I LLC and Intellectual Ventures II LLC

Example American Count VI Systems and Services
U.S. Patent No. 7,721,282 ("the '282 Patent")

The Accused Systems and Services include without limitation American systems and services that utilize Docker; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future American systems and services that have the same or substantially similar features as the specifically identified systems and services ("Example American Count VI Systems and Services" or "American Systems and Services").[1]

On information and belief, the American Systems and Services use Docker in public and/or private cloud(s). For example, American posts, or has posted, job opportunities that require familiarity with Docker containerization concepts.

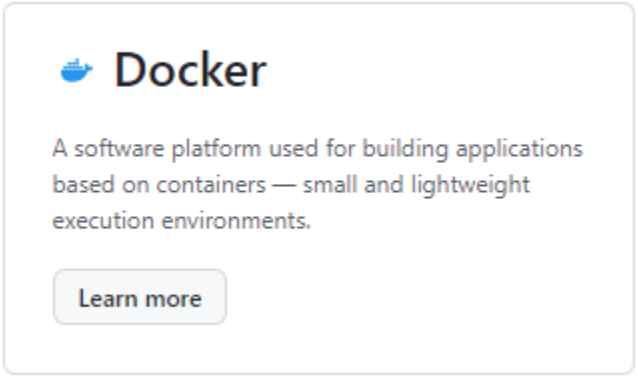
- Example of job posting for an Engineer/Sr Engineer in IT Situational Awareness at American Airlines which mentions Docker as a necessary skill for the position. https://jobs.aa.com/job/EngineerSr-Engineer%2C-IT-Situational-Awareness/75837-en_US. (Last accessed on 10/31/2024).
- Example of job posting for an Associate Developer in IT Applications at American Airlines which mentions Docker as a necessary skill for the position. https://jobs.aa.com/job/Associate-Developer%2C-IT-Applications/75816-en_US. (Last accessed on 10/31/2024).
- Example of Sr. Cloud Infra DevSecOps Engineer/Architect position at American Airlines which mentions use of Docker. https://www.linkedin.com/in/rupa-m-b90836309/. (Last accessed on 9/19/24).
- Example of Kubernetes Engineer position at American Airlines which mentions use of Docker. https://www.linkedin.com/in/sridhar-pulluri-199b56250/. (Last accessed on 9/19/24)
- Example of Software Engineer position at American Airlines which mentions use of Docker. https://www.linkedin.com/in/pthotakura9/. (Last accessed on 9/19/24)
- Example of Senior Developer, IT Applications position at American Airlines which mentions use of Docker. https://www.linkedin.com/in/cj-cohorst-61a614173/. (Last accessed on 9/19/24)

[1] Plaintiffs do not accuse the public clouds of Defendant, to the extent those services are provided by a cloud provider with a license to Plaintiffs' patents that covers Defendant's activities. Plaintiffs do not accuse the public clouds of Defendants if those services are provided by a cloud provider with a license to Plaintiffs' patents that covers Defendants' activities. Plaintiffs will produce relevant license agreements in this litigation. Plaintiffs accuse Defendant private clouds that implement Docker and non-licensed public clouds that Defendant uses to support Docker for its systems and services. Plaintiffs will provide relevant license agreements for cloud providers in discovery, to the extent any such license agreements have not already been produced. To the extent any of these licenses are relevant to Defendant's activities, Plaintiffs will meet and confer with Defendant about the impact of such license(s). Once a protective order is entered into the case, Plaintiffs will provide further details.

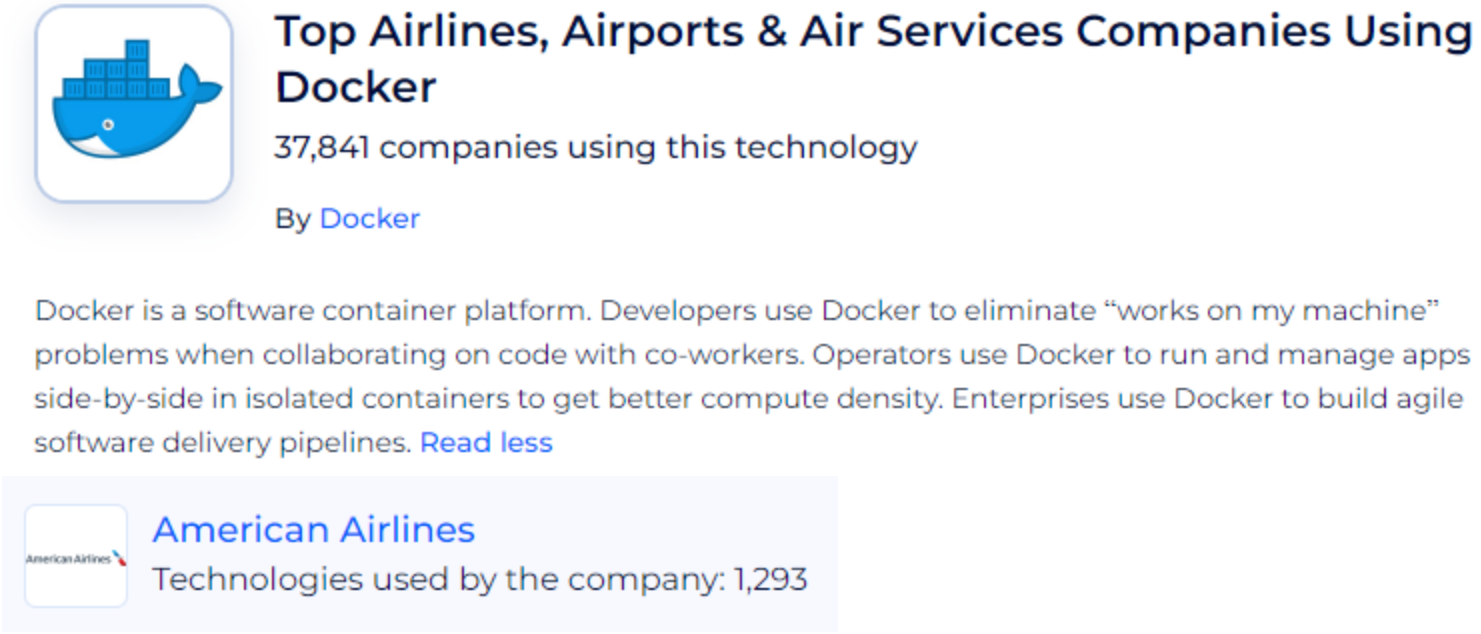
- Example of Developer position at American Airlines which mentions use of Docker. https://in.linkedin.com/in/prudhvi-kumar-rayapati.

As another example, American has announced cloud migration of legacy technology and efforts to modernize its mainframes and servers. Source: https://dxc.com/sg/en/insights/customer-stories/american-airlines-cloud-data-automation. American continues to use private cloud for at least certain applications. Source: https://www.techtarget.com/searchdatamanagement/feature/American-Airlines-lowers-data-management-costs-with-Intel. ("American Airlines' initial target for cost optimization was Azure Data Lake, according to Vijay Premkumar, senior manager of public and *private cloud* at the airline.") (emphasis added).

On information and belief, other information confirmed American uses Docker technology.



Source: https://github.com/orgs/americanair/packages.



Source: https://www.zoominfo.com/tech/23717/docker-tech-from-transportation-airline-industry-in-us-by-revenue.

When I came to American Airlines, I tried to contribute to several InnerSource projects in our corporate VCS. One of the things I observed is that local development leaned on local installations of the application framework extensively. In the past, I've been victim of "worked on my machine" in similar setups and wanted to get a better understanding of how pervasive containers were being used. I wanted to present options on how to use containers to help remove local dependency hell. There was plenty of opportunity to leverage this approach for local development 🤼

Using containers for a local development environment can help remove impediments in situations such as:

- My stack has several dependencies that are hard to emulate:
  - Databases, and associated volumes for (short-lived) storage
  - Caches
  - Connectivity to the above, with declarative service names
- I have a bunch of application dependencies (`requirements.txt`, `packages.json`, `Gemfile`)
- There are many moving parts to reproduce a production-like environment
- My team uses different operating systems, or versions of operating systems

Windows and macOS

- Install Docker - `stable` or `edge` is fine. For the purposes of this write-up, we're using basic features.

Source: https://tech.aa.com/2020-10-13-containers-local-dev/.

Karl Haworth, is a Principal Staff Engineer on American Airlines Developer Experience products. His team is leveraging Chainguard Images to harden their software and strengthen the security of their software supply chain. Karl recently created an alternative, secure image for the Backstage open-source framework using Changuard's wolfi-base image. In this guest post, Karl explores the decision to use wolfi-base and the benefits he has seen in reducing vulnerabilities and shrinking the overall Backstage attack surface.

American Airlines is committed to enhancing the developer experience by implementing a frictionless self-service platform to create delightful developer experiences. By doing so, the developers can deliver value *sooner* to our customers. In order to achieve the stated goal, we adopted Spotify's Backstage open-source framework for accelerating the development of our internal developer portal Runway. This marks my third endeavor at establishing a developer portal which has been successful due to a heavy focus on InnerSource practices. Backstage has proven instrumental in streamlining the development process, thanks to its community-driven foundation as a solid base to build upon.

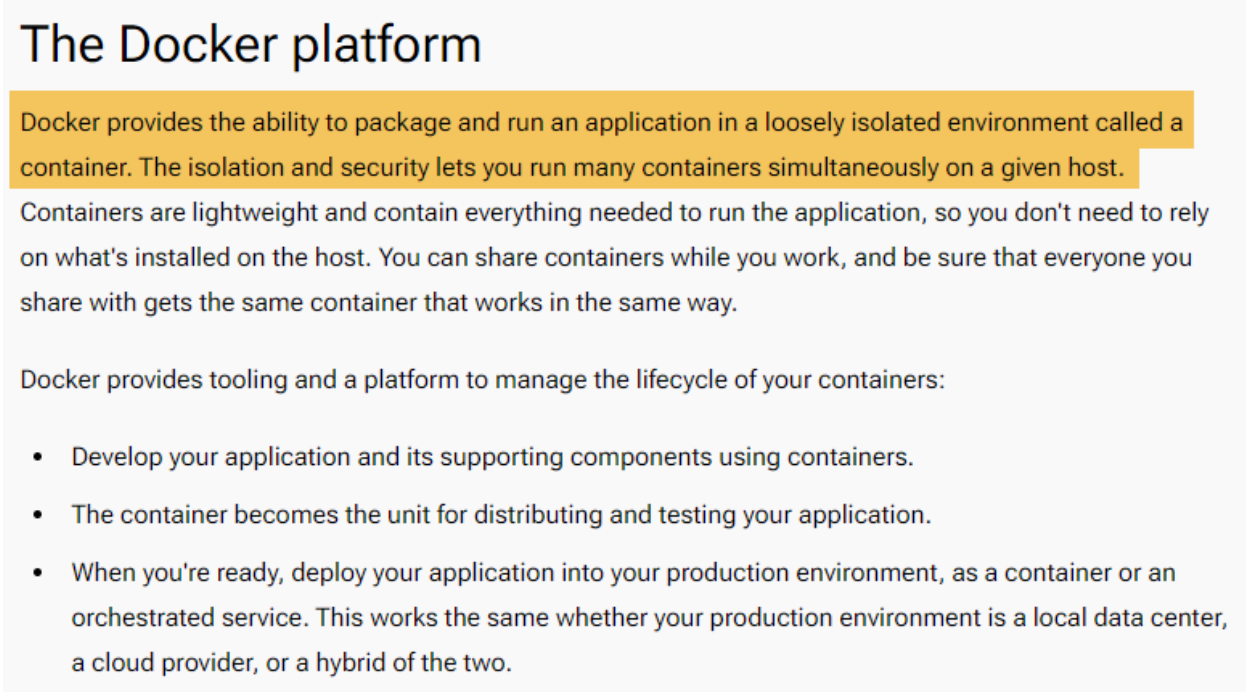
While Backstage prioritizes security, concerns arise when utilizing the base image, `node:18-bookworm-slim`, as it contains 74 vulnerabilities ranging in severity. This has prompted us to assess the security implications when using the docker build command and `Dockerfile` within the Backstage framework, as the default added an additional 330 vulnerabilities. I attempted using updated node and Debian images, but they also yielded similar results, prompting me to seek out different solutions.

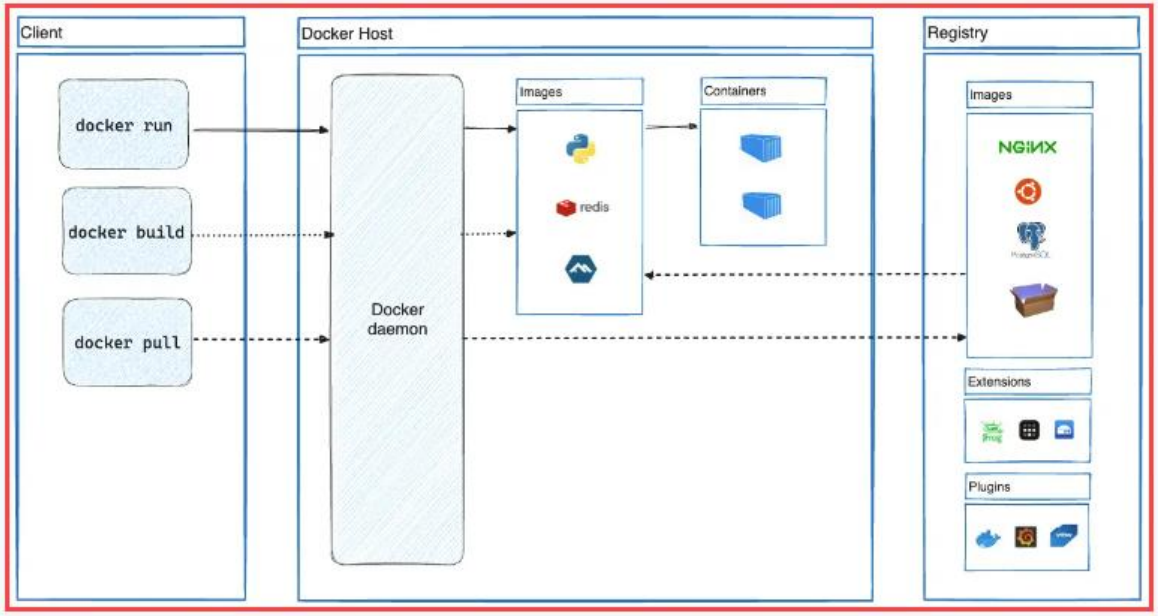
Source: https://www.chainguard.dev/unchained/reducing-vulnerabilities-in-backstage-with-chainguards-wolfi. *See also* https://karlhaworth.com/.[2]

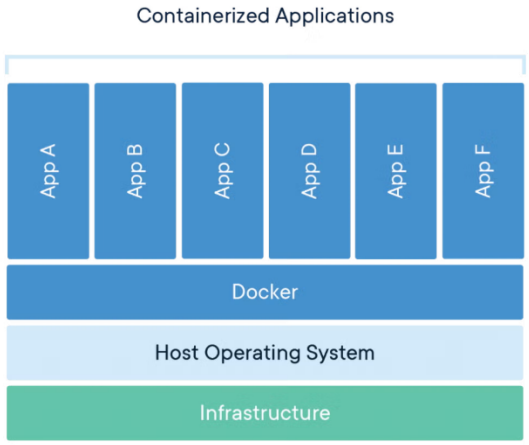
[2] Unless otherwise noted, all sources cited in this document were publicly accessible as of the filing date of the Complaint.

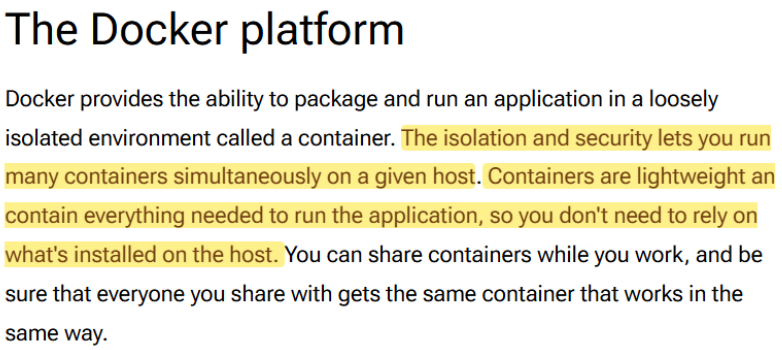
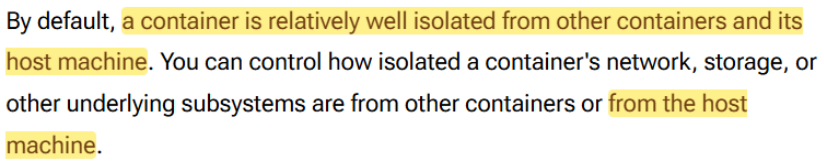
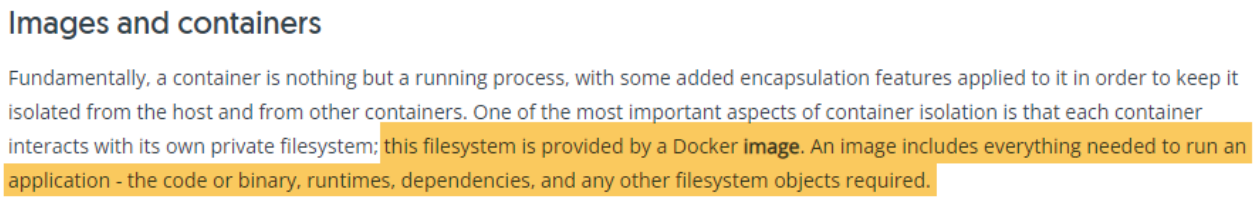
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
1. A system for distributing an application environment comprising:	To the extent this preamble is limiting, on information and belief, the American Count VI Systems and Services are a "system for distributing an application environment." Docker is an open-source platform that enables the development and distribution of applications and facilitates running applications in an isolated environment called containers. A multi-server environment capable of deploying and running Docker containers is considered a system. **Docker overview** Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production. Source: https://docs.docker.com/guides/docker-overview/.[3]

[3] Annotations added unless otherwise noted.

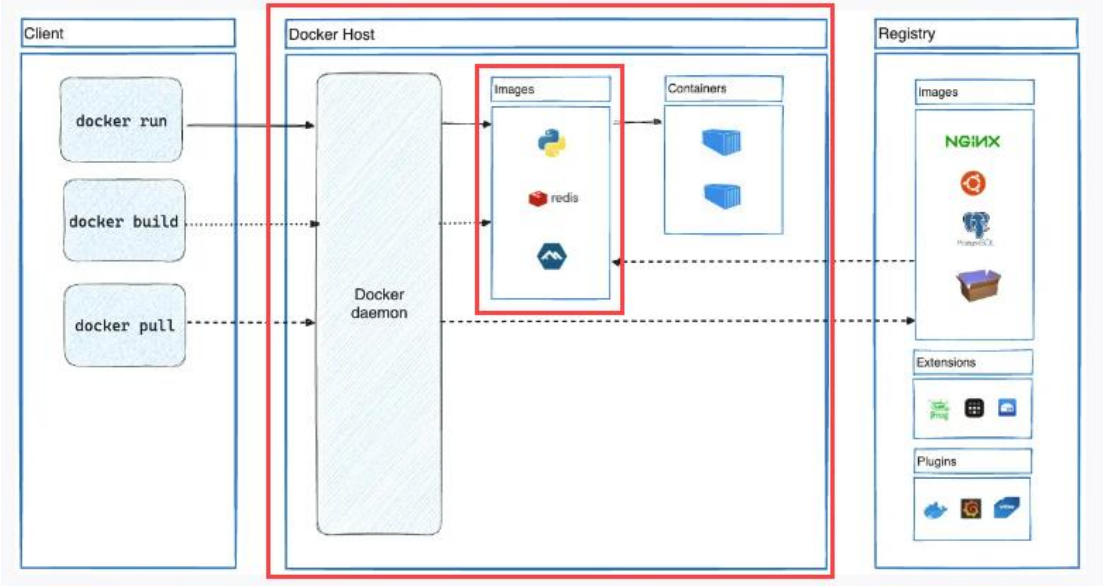
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/guides/docker-overview/. For example, a user can interact with Docker on the command line to run, build, pull, and/or configure containers, images, and/or volumes.

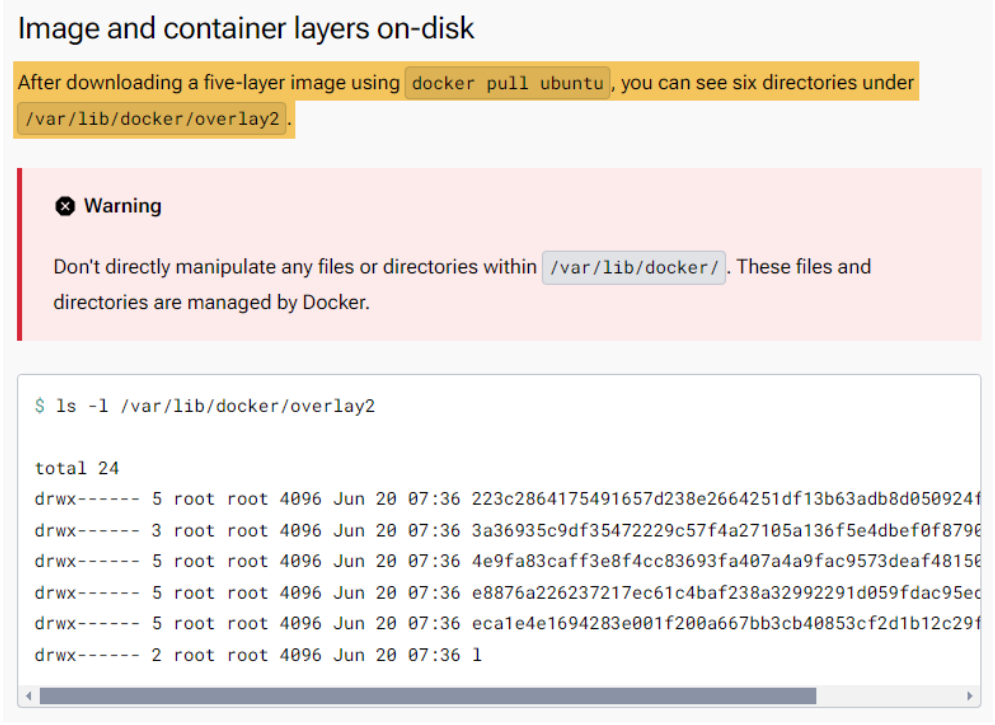
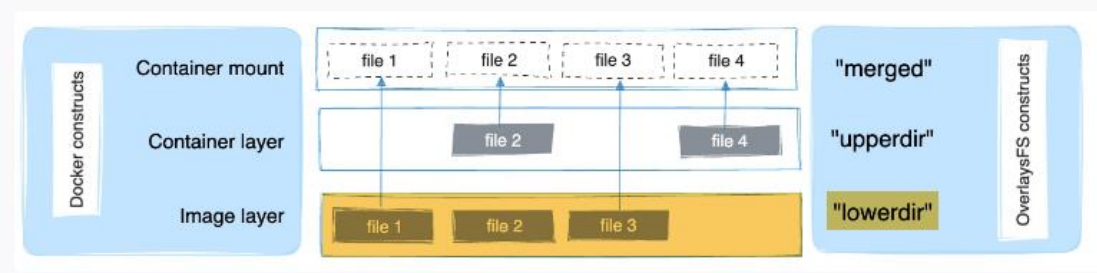
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/guides/docker-overview/.
1[a] a compute node comprising a computer system;	On information and belief, the American Count VI Systems and Services include "a compute node comprising a computer system." Docker requires a host environment or a server to host containers in isolation from each other. Containers (e.g., a compute node) are running instances of an image, having executable packages of software, including code, runtime, system tools, libraries, and settings (e.g., comprising a computer system).

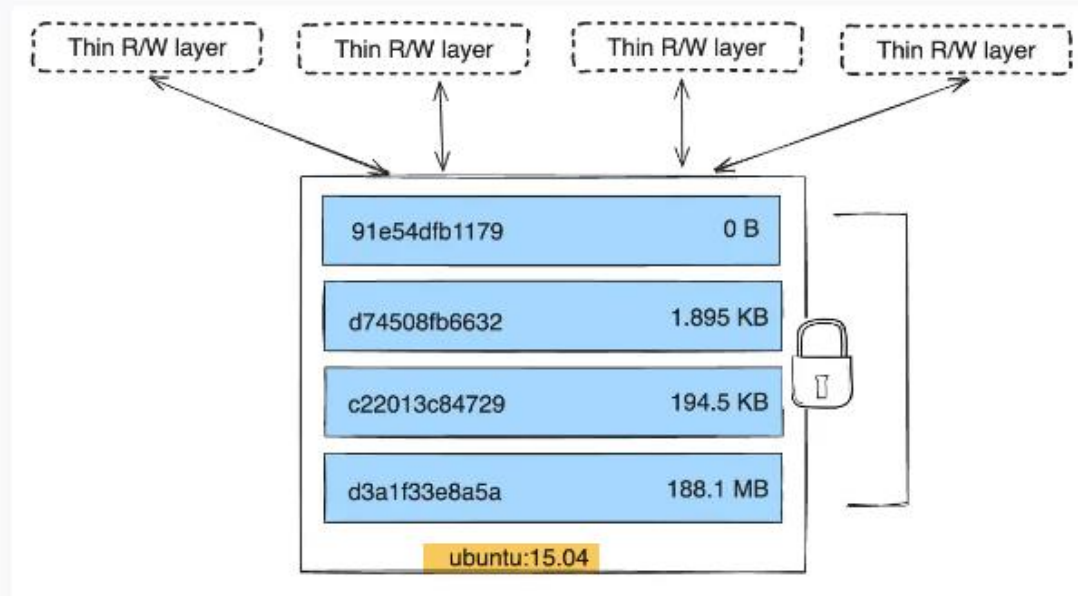
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Source: https://www.docker.com/resources/what-container/.  Source: https://www.docker.com/resources/what-container/.

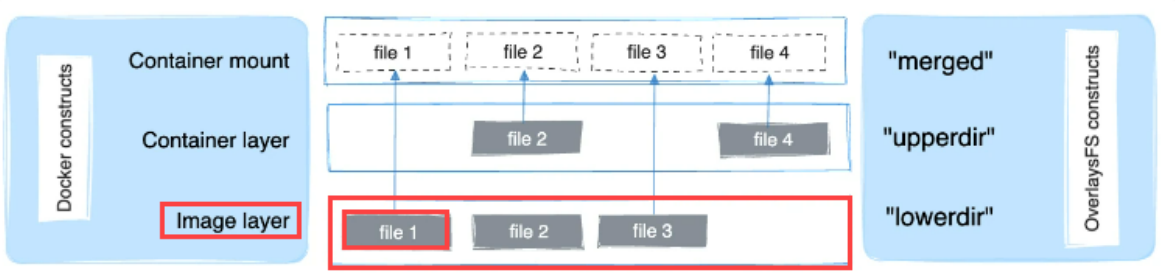
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/get-started/docker-overview/.  Source: https://docs.docker.com/get-started/docker-overview/. For example, containers are running instances of images. Docker images include the libraries, dependencies, and other environment elements for an application to run in. The storage driver controls how the images and container instances are resident on the host.  Source: https://docker-docs.uclv.cu/get-started/.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	Docker supports several storage drivers, using a pluggable architecture. The storage driver controls how images and containers are stored and managed on your Docker host. After you have read the storage driver overview, the next step is to choose the best storage driver for your workloads. Use the storage driver with the best overall performance and stability in the most usual scenarios. Source: https://docs.docker.com/storage/storagedriver/select-storage-driver/.
1[b] a first storage unit for storing blocks of a root image of the compute node, wherein the first storage unit comprises a first non-volatile memory, wherein the root image comprises a computer program, wherein the blocks comprise sections of data, and wherein a file of the root image comprises at least one block;	On information and belief, the American Count VI Systems and Services include "a first storage unit for storing blocks of a root image of the compute node, wherein the first storage unit comprises a first non-volatile memory, wherein the root image comprises a computer program, wherein the blocks comprise sections of data, and wherein a file of the root image comprises at least one block." Docker containers each have their own private filesystem provided by a Docker image. Docker provides storage drivers that control how the images and containers are stored and managed on a Docker host. Docker supports several storage drivers, using a pluggable architecture. The storage driver controls how images and containers are stored and managed on your Docker host. After you have read the storage driver overview, the next step is to choose the best storage driver for your workloads. Use the storage driver with the best overall performance and stability in the most usual scenarios. Source: https://docs.docker.com/storage/storagedriver/select-storage-driver/.

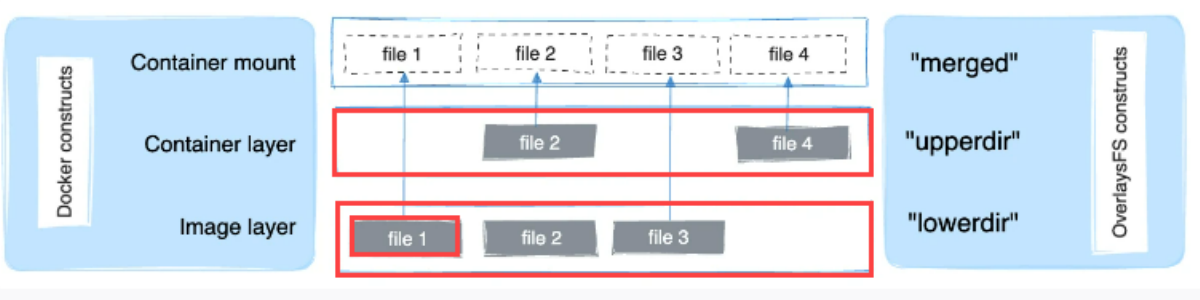
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/guides/docker-overview/. For example, Docker stores images and containers in memory on the host system. Particularly, Docker, through its OverlayFS (Overlay File System), presents all the directories as a single consolidated directory. The lower directory is called 'lowerdir' and the upper directory is called 'upperdir'. The 'lowerdir' is the layer that stores blocks of the base or root image. The base or root image includes the application code, dependencies, and other necessary elements for execution or runtime. Each layer, or the files therein, occupies a portion of the 'lowerdir'.

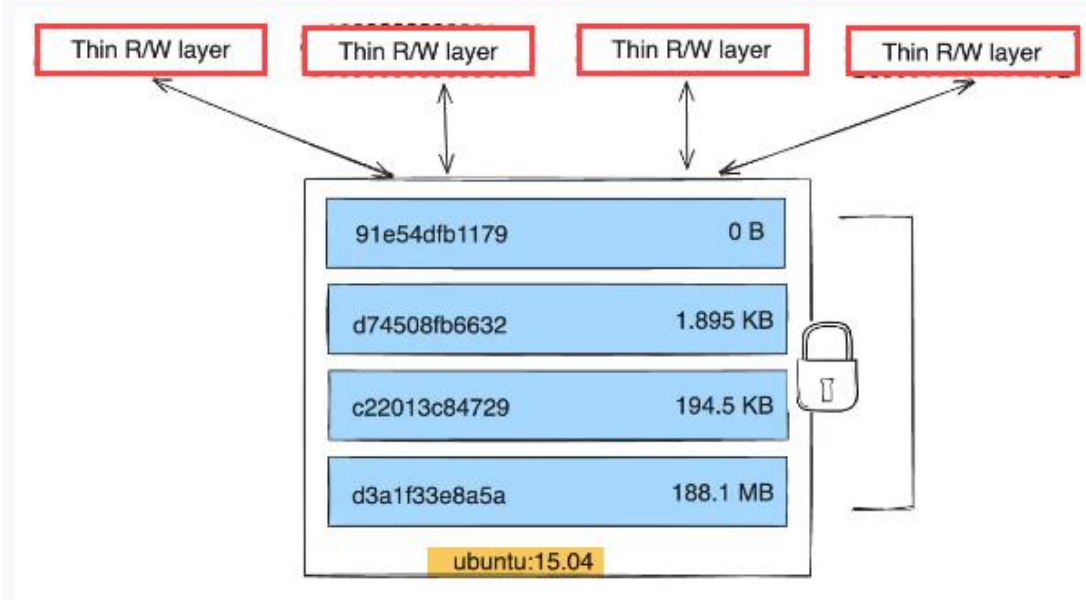
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/.  Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/storage/storagedriver/. As shown in the evidence below, each layer can include multiple files. For example, an image layer can include three files, where each of these files includes at least one block of data. A user can see a consolidated view via the merged directory, which is a combined view of the 'lowerdir' and 'upperdir' via overlay2, the storage driver in OverlayFS.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	The following diagram shows how a Docker image and a Docker container are layered. The image layer is the `lowerdir` and the container layer is the `upperdir`. If the image has multiple layers, multiple `lowerdir` directories are used. The unified view is exposed through a directory called `merged` which is effectively the containers mount point.  Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/. Image and container layers on-disk After downloading a five-layer image using `docker pull ubuntu`, you can see six directories under `/var/lib/docker/overlay2`. Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/.
1[c] a second storage unit for storing a leaf image, the leaf image comprising new data blocks and changes to the blocks of the root image, wherein the second storage	On information and belief, the American Count VI Systems and Services include "a first storage unit for storing blocks of a root image of the compute node, wherein the first storage unit comprises a first non-volatile memory, wherein the root image comprises a computer program, wherein the blocks comprise sections of data, and wherein a file of the root image comprises at least one block." Docker includes a top writable layer, where all writes to a container (e.g., a write to add new data or modify existing data) are stored in the writable layer. The writable layer contains only the modified data

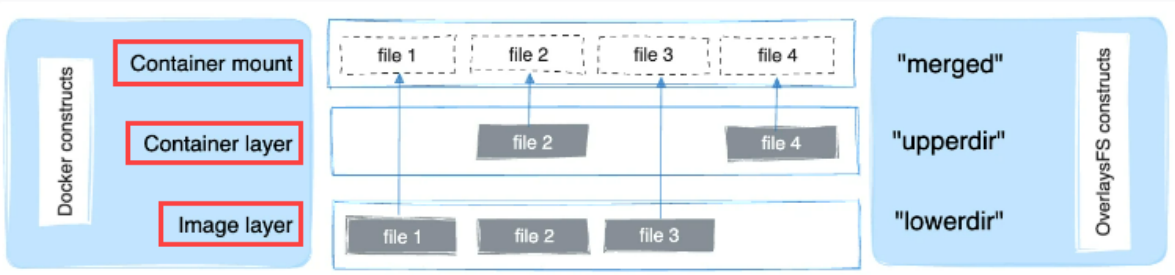
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
unit comprises a second non-volatile memory; and	per the changes made by the respective container instance. The writable layer is stored in memory. Container and layers The major difference between a container and an image is the top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged. Source: https://docs.docker.com/storage/storagedriver/. Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image. Source: https://docs.docker.com/storage/storagedriver/. For example, the changes, edits, or modifications made to a container will be stored in the ‘upperdir’ of that container. The ‘upperdir’ includes only the changed, edited, or modified file(s) by the container.

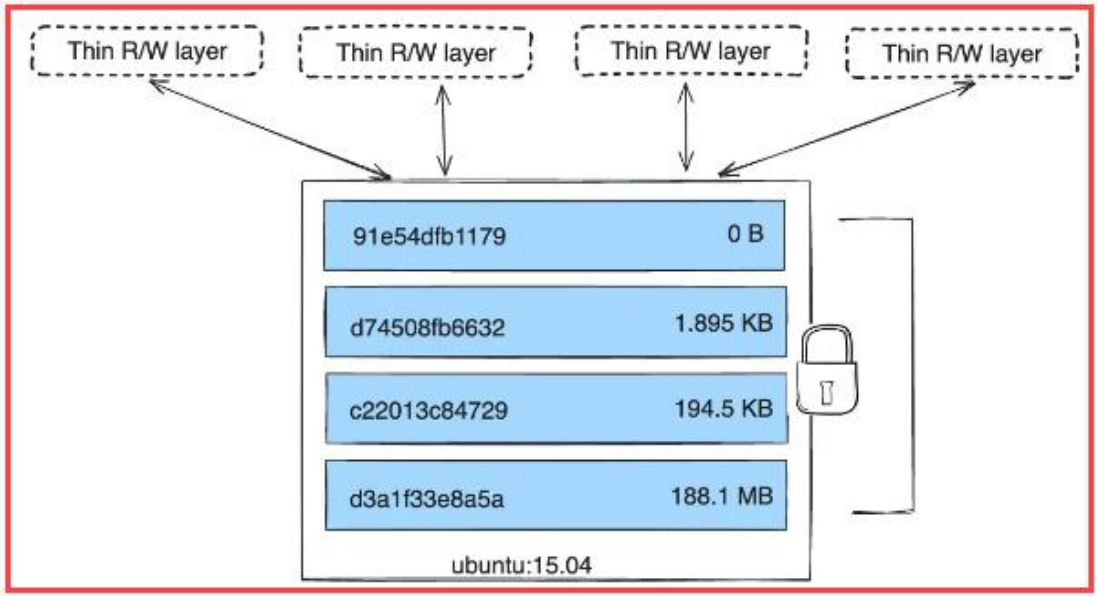
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	The following diagram shows how a Docker image and a Docker container are layered. The image layer is the `lowerdir` and the container layer is the `upperdir`. If the image has multiple layers, multiple `lowerdir` directories are used. The unified view is exposed through a directory called `merged` which is effectively the containers mount point.  Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/.

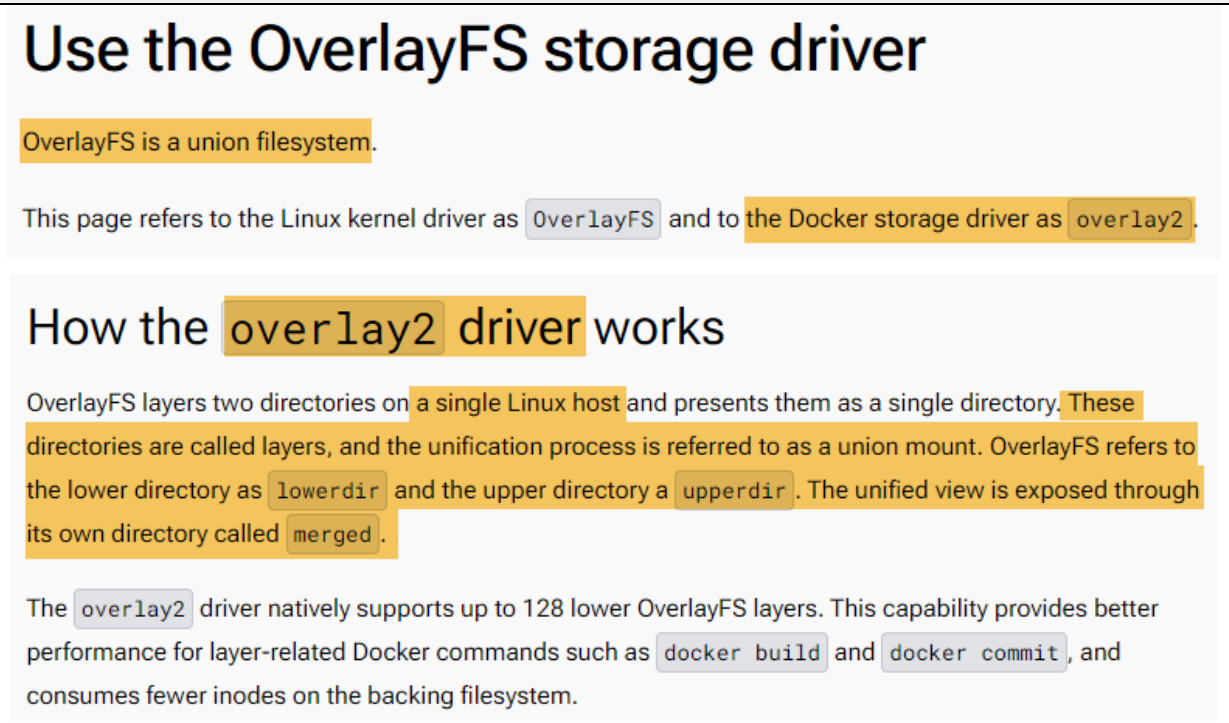
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/storage/storagedriver/. For example, below describes how overlay2 functions by comparing and calculating the differences or changes between the parent and its corresponding layer, if it was modified.

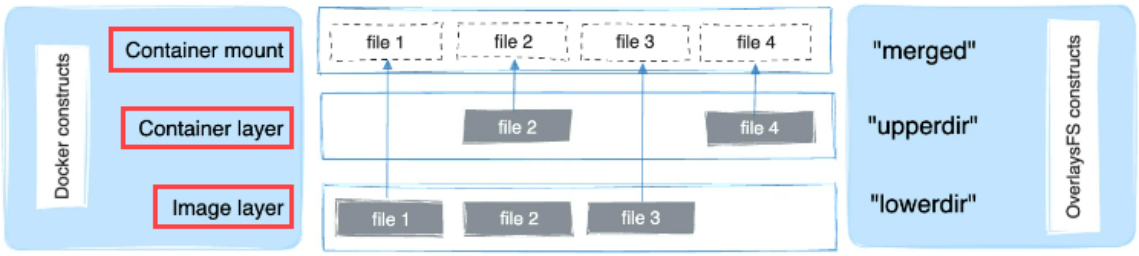
U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	`` // DiffSize calculates the changes between the specified id // and its parent and returns the size in bytes of the changes // relative to its base filesystem directory. func (d *Driver) DiffSize(id, parent string) (int64, error) { if useNaiveDiff(d.home) || !d.isParent(id, parent) { return d.naiveDiff.DiffSize(id, parent) } return directory.Size(context.TODO(), d.getDiffPath(id)) } // Diff produces an archive of the changes between the specified // layer and its parent layer which may be "". func (d *Driver) Diff(id, parent string) (io.ReadCloser, error) { if useNaiveDiff(d.home) || !d.isParent(id, parent) { return d.naiveDiff.Diff(id, parent) } // Changes produces a list of changes between the specified layer and its // parent layer. If parent is "", then all changes will be ADD changes. func (d *Driver) Changes(id, parent string) ([]archive.Change, error) { return d.naiveDiff.Changes(id, parent) } ``` Source: https://github.com/moby/moby/blob/master/daemon/graphdriver/overlay2/overlay.go.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
1[d] a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, wherein the union block device comprises a driver, wherein the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit; the union block device comprises a low-level driver for interfacing between the first and second storage units and the file system of the compute node; and the union block device, upon receiving a write request from the compute node for a sector X, creates an appropriate persistent mapping for sector X.	On information and belief, the American Count VI Systems and Services include "a union block device for interfacing between the compute node and the first and second storage units to distribute the application environment to the compute node, wherein the union block device comprises a driver, wherein the union block device creates the application environment by merging the blocks of the root image stored on the first storage unit with the blocks of the leaf image stored on the second storage unit; the union block device comprises a low-level driver for interfacing between the first and second storage units and the file system of the compute node; and the union block device, upon receiving a write request from the compute node for a sector X, creates an appropriate persistent mapping for sector X." Docker enables building, running, and sharing applications using containers. Docker provides a storage driver – overlay2, for managing and storing images for use. The overlay2 storage driver provides a mechanism for interfacing between the container and storage units. **How the `overlay2` driver works** OverlayFS layers two directories on a single Linux host and presents them as a single directory. These directories are called layers, and the unification process is referred to as a union mount. OverlayFS refers to the lower directory as `lowerdir` and the upper directory a `upperdir`. The unified view is exposed through its own directory called `merged`. The `overlay2` driver natively supports up to 128 lower OverlayFS layers. This capability provides better performance for layer-related Docker commands such as `docker build` and `docker commit`, and consumes fewer inodes on the backing filesystem. Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	The following diagram shows how a Docker image and a Docker container are layered. The image layer is the `lowerdir` and the container layer is the `upperdir`. If the image has multiple layers, multiple `lowerdir` directories are used. The unified view is exposed through a directory called `merged` which is effectively the containers mount point.  Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/. To create a container, the `overlay2` driver combines the directory representing the image's top layer plus a new directory for the container. The image's layers are the `lowerdirs` in the overlay and are read-only. The new directory for the container is the `upperdir` and is writable. Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/storage/storagedriver/. **Images and containers** Fundamentally, a container is nothing but a running process, with some added encapsulation features applied to it in order to keep it isolated from the host and from other containers. One of the most important aspects of container isolation is that each container interacts with its own private filesystem; this filesystem is provided by a Docker **image**. An image includes everything needed to run an application - the code or binary, runtimes, dependencies, and any other filesystem objects required. Source: https://docker-docs.uclv.cu/get-started/. Docker includes an OverlayFS union filesystem that includes a storage driver, overlay2. OverlayFS includes the image layer and the container layer, which are presented as a single directory in a process called a union mount. In a container filesystem, the overlay2 storage driver combines directories representing the image's read-only layer and writable layers.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	 Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/. For example, Docker refers to the base image layers as 'lowerdir' and the container layer as 'upperdir,' and the unified view is referred to as merged.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	The following diagram shows how a Docker image and a Docker container are layered. The image layer is the `lowerdir` and the container layer is the `upperdir`. If the image has multiple layers, multiple `lowerdir` directories are used. The unified view is exposed through a directory called `merged` which is effectively the containers mount point.  Source: https://docs.docker.com/storage/storagedriver/overlayfs-driver/. Furthermore, all writes are made in the top writable layer (container layer / 'upperdir'). Based on information and belief, it is assumed that writes are made in a specific portion of the writable layer, particularly in the case of modifications made to the existing data stored in an already known portion of the writable layer. **Container and layers** The major difference between a container and an image is the top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged. Source: https://docs.docker.com/storage/storagedriver/.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	The second-lowest layer, and each higher layer, contain a file called `lower`, which denotes its parent, and a directory called `diff` which contains its contents. It also contains a `merged` directory, which contains the unified contents of its parent layer and itself, and a `work` directory which is used internally by OverlayFS. Source: https://docs.docker.com/engine/storage/drivers/overlayfs-driver/. The first time a container writes to an existing file, that file does not exist in the container (`upperdir`). The `overlay2` driver performs a `copy_up` operation to copy the file from the image (`lowerdir`) to the container (`upperdir`). The container then writes the changes to the new copy of the file in the container layer. However, OverlayFS works at the file level rather than the block level. This means that all OverlayFS `copy_up` operations copy the entire file, even if the file is large and only a small part of it's being modified. This can have a noticeable impact on container write performance. However, two things are worth noting: Source: https://docs.docker.com/engine/storage/drivers/overlayfs-driver/. For example, the 'lowerdir' directory can be seen by the *getLowerDirs* function to return all the files associated with the parent or base image.

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	468 `func (d *Driver) getLowerDirs(id string) ([]string, error) {` 469 `var lowersArray []string` 470 `lowers, err := os.ReadFile(path.Join(d.dir(id), lowerFile))` 471 `if err == nil {` 472 `for _, s := range strings.Split(string(lowers), ":") {` 473 `lp, err := os.Readlink(path.Join(d.home, s))` 474 `if err != nil {` 475 `return nil, err` 476 `}` 477 `lowersArray = append(lowersArray, path.Clean(path.Join(d.home, linkDir, lp)))` 478 `}` 479 `} else if !os.IsNotExist(err) {` 480 `return nil, err` 481 `}` 482 `return lowersArray, nil` Source: https://github.com/moby/moby/blob/master/daemon/graphdriver/overlay2/overlay.go. Furthermore, overlay2 first looks at if there have been any modifications at the writable layer. It will then merge the lower and upper directories and return that as the mount path of the container. For example, within a container, by listing out the files. 508 `// Get creates and mounts the required file system for the given id and returns the mount path.` 509 `func (d *Driver) Get(id, mountLabel string) (_ string, retErr error) {`

U.S. Patent No. 7,721,282 (Claim 1)	
Claim	**Example American Count VI Systems and Services**
	`` 527 mergedDir := path.Join(dir, mergedDirName) 528 if count := d.ctr.Increment(mergedDir); count > 1 { 529 return mergedDir, nil 530 } 531 defer func() { 532 if retErr != nil { 533 if c := d.ctr.Decrement(mergedDir); c <= 0 { 534 if mntErr := unix.Unmount(mergedDir, 0); mntErr != nil { 535 logger.Errorf("error unmounting %v: %v", mergedDir, mntErr) 536 } 537 // Cleanup the created merged directory; see the comment in Put's rmdir 538 if rmErr := unix.Rmdir(mergedDir); rmErr != nil && !os.IsNotExist(rmErr) { 539 logger.Debugf("Failed to remove %s: %v: %v", id, rmErr, err) 540 } 541 } 542 } 543 }() 565 mountData := label.FormatMountLabel(opts, mountLabel) 566 mount := unix.Mount 567 mountTarget := mergedDir ``` Source: https://github.com/moby/moby/blob/master/daemon/graphdriver/overlay2/overlay.go.